REMARKS

Claims 1, 2, 5-14, 16-18 and 21-24 are now pending in this application. Claims 12-14, 16 and 17 have been objected to for minor informalities. Claims 1, 2, 5-8, 10 and 11 stand rejected under 35 U.S.C. § 103(a) as being allegedly being anticipated by United States Patent 6,003,095 ("Pekowski02") in view of United States Patent 5,946,486 ("Pekowski01) and in further view of United States Patent 6,226,618 ("Downs"). Applicants respectfully traverse.

Claims 1-2, 5-12, 18 and 23-24 have been amended. Claim 26 has been added.

Objections Relating To Claims 12-14, 16 and 17

Claim 14 has been amended to change the term "software module" to "program module". Thus, the objections relating to claim 12 should be withdrawn. Claims 13-14 and 16-17 depend from and therefore include all the limitations of claim 12. Thus, the objections to claims 13-14 and 16-17 should also be withdrawn.

Rejection Of Claims 1-2 and 5-11 Under 35 U.S.C. § 103(a)

Claims 1-2 and 5-11 stand rejected under 35 U.S.C. § 103(a) as being allegedly being anticipated by Pekowski02 in view of Pekowski01 and in further view of Downs. Claim 1 as amended recites in a runtime environment comprising a first program module, at least one second program module and a call stack, a method of invoking a desired method associated with a desired second program module comprising *in a third program module associating each of a plurality of stubs respectively with each of a plurality of methods associated with the second program module, wherein each stub comprises a code segment performing a unique non-standard calling convention into the second program module, wherein each stub includes at least a first instruction to push function parameters onto the call stack, a second instruction to call an authenticator module for authenticating that a stub has not been modified and a third instruction comprising embedded unique data for the stub, wherein the embedded data comprises a vtable entry descriptor for the desired method, corresponding to a vtable for the third module, wherein the vtable is covered and comprises a list of function pointers to functions associated with the second module arranged in a random order, the*

random order unique for each second module.

Claim 1 as amended further *recites from the first program module, issuing a first call to a stub in the third program module associated with the desired method, whereupon after the first call, the call stack comprises at least a first parameter corresponding to a return address associated with the stub, a second parameter corresponding to a parameter depth (cArgs) and a third parameter corresponding to a return address of the first program module, the first, second and third parameters arranged in a top-down order.*

Claim 1 as amended further recites *wherein the third program module calls the second program module using a non-standard calling convention.*

Support for this amendment may be found, for example, on pages 9-22 of the specification. Referring to FIG. 5 of the specification, program module 136 may be a dynamic-link library (DLL) that performs sensitive functions that should be performed under controlled circumstances (e.g., decryption of valuable information). Application 135 (depicted, by way of example, as a "3rd party client application") may be a software application that calls on program module 136 to perform such functions. (Program module 136, in the example of FIG. 5, is a "black box"). Application 135, however, does not call program module 136 directly, but rather calls program module 136 through an intermediate module 502. In the example of FIG. 5, intermediate module 502 is a DLL named "msdrm.dll," which exposes an application programming interface (API) that provides DRM-related functions to applications. Program module 502 includes functionality called "black box stubs," which are the code segments that are callable by application 135 and are used by intermediate module 502 to call into the black box. Preferably, the stubs are carefully contrived hand-coded assembly language stubs that perform a unique calling convention into the black box; such a calling convention is more particularly described below. (It should be understood that the references to DRM functionality and to the black box are merely exemplary, and the structure of FIG. 5 is not limited to DRM systems.)

Program module 136 is not exposed directly to application 135. Intermediate module 502 may use a non-standard calling convention to call program module 136.

In the example of FIG. 5, program module 502 is exposed to application 135, but program module 136 is not exposed to application 135. However, program module 502 contains "stubs" that can be used to enter program module 136 - i.e., application 135 invokes

the stubs, and this invocation identifies the particular functionality from program module 136 that application 135 would like to perform. The stubs, in turn, may invoke the relevant code in program module 136. Program module 136, in this example, includes the code need to verify and authenticate the caller (i.e., application 135), and the stubs contain data required by the verification and authentication code. Preferably, this data is mixed into the stubs' instruction streams. The call into program module 136 passes the address of this data to the verification and authentication code "surreptitiously." That is, data used by the verification and authentication code is inserted into the instruction stream immediately after the call to program module 136; since the call causes the address immediately following the call instruction to be pushed onto the stack as a return address, the stack contains a pointer to the data in what would be the return address using a standard calling convention. If a standard tool were to patch the stub, then the return address will no longer point to the actual data that is needed by the verification function. The address of the stub itself is also preserved and the entire stub is verified by the code that ultimately executes the call (or jump) into the code providing the requested functionality, for example a content decryption request, in program module 136.

On the x86 processor architecture, the DRMxxxStub may look like (in pseudo-assembly):

```
push cArgs                    ;number of DRMxxx function parameters
call dword ptr [g_pfnBBJump]  ;call blackbox "demux" authenticator
Ox07                          ;data injected in instruction stream
```

The return address pushed on to the stack as a result of the "call" will actually point at the "0x07" data. The blackbox demux authenticator will not actually return to that location, but will simply use the data and return directly to the calling application instead. This prevents standard pro filers and binary patching tools from adding logging or redirection code at this interface, and prevents debuggers from adding breakpoints. It should be noted that data such as "0x07" may be different for each stub and will be interpreted as a random instruction opcode by disassemblers (in this case, "pop e s "), which is also a benefit. Since most instructions are longer than one byte, the bogus opcode will cause further mistakes in the disassembly as parts of other stubs are interpreted as part of that instruction (i.e. the disassembler gets "out of alignment").

Referring to the exemplary call stack layout upon entering BBJump, shown on page 15 of the specification, BBJump uses the DRMxxx Stub return address to verify the stub has not been modified and to access the embedded data, which is actually a vtable entry descriptor for the desired DRM service. This descriptor and the "Parameter Depth" are used by the demux to call the desired function, DRMxxxPub, inside the blackbox and to prepare the stack for return to the application (i.e. resetting the stack to adhere to stdcall calling convention). The vtable is a covered (i.e. encrypted) list of function pointers inside the binary, in random order (i.e. encrypted and ordered uniquely for each blackbox).
[0051] After verification, the application return address is inserted in the stack below the DRMxxx parameters, so that BBJump will return directly to the application rather than the stub. The DRMxxx Stub Return Address is replaced on the stack with the address of the DRMxxxPub address, and the cArgs is replaced with a return address inside BBJump for cleanup.

Page 16 of the call stack shows the stack after these replacements have been performed. By doing this, the subsequent jump into the unique v-table uncovering code will transition into the DRMxxxPub function when it attempts to "return" to BBJump. That is, the address of the DRMxxxPub function is surreptitiously used by BBJump to force the blackbox to transition into the DRMxxxPub function without returning to BBJump and making another call. Since the BBJump Cleanup address has been inserted below that, DRMxxxPub will return directly to BBJump for cleanup, which includes obliterating sensitive data from the recently used parts of the stack.

One feature provided herein is to authenticate the code that calls a program module - i.e., to verify that the code surrounding the call has not been modified relative to a known state. Module authentication is a security feature that helps protect the application execution environment against tampering. There are two phases to module authentication. Static authentication is a one-time check of module on-disk images while dynamic authentication is a check of the in-memory module images. Dynamic authentication is ongoing process since tampering can occur anytime in the life of the application process.

Pekowski02 relates to a method for automatically generating the code necessary to implement an operating system's demand loading APIs. The resulting code is packaged into a library, which replaces a statically linked DLL reference library. In this manner, the called

DLL is not changed in any way, but a tool is run against it to automatically generate the demand loading library.

Pekowski02 describes a method, implemented in an information handling system having an operating system, for demand loading a DLL having a reference library for resolving references to the DLL. According to Pekowski02 the method comprises the steps of generating a demand load library for demand loading the DLL and replacing the reference library of the DLL with the demand load library. The step of generating a demand load library comprises the steps of finding all entry points into the DLL and generating program code for performing the following steps for each entry point: determining whether an address corresponding to the entry point has been called before; if the address corresponding to the entry point has not been called before, then calling a demand loading initialization routine; and jumping indirectly to the address corresponding to the entry point. The step of calling a demand loading initialization routine includes the step of calling APIs from the operating system.

Pekowski02 also describes a method, implemented in an information handling system having an operating system, for generating a demand load library for demand loading a DLL. The method comprises the steps of finding all entry points into the DLL; and generating program code for performing the following steps for each entry point: determining whether an address corresponding to the entry point has been called before; if the address corresponding to the entry point has not been called before, then calling a demand loading initialization routine; and jumping to the address corresponding to the entry point.

Pekowski01 relates to a method and apparatus for automatically tracing new code in a software module without accessing or modifying the source code of the software module. Pekowski01 describes a method, implemented in an information handling system, for tracing events occurring upon entries to or exits from a named software module. The method comprises the steps of providing a software module having a name; generating a shadow software module having the same name as the software module for tracing events occurring upon entry to or exit from the software module and renaming the software module, thereby avoiding modifications to the software module.

Pekowski01 also describes a method, implemented in an information handling system, for generating a shadow software module for tracing events occurring upon entry to and exit

from a software module. The method comprises the steps of generating program code for an entry hook routine for performing a common entry processing function; and generating program code for an exit hook routine for performing a common exit processing function.

Pekowski01 further describes a method, implemented in an information handling system, for generating a shadow software module for tracing events occurring upon entry to or exit from a named software module. The method comprises the steps of inputting the named software module; renaming the software module; generating a shadow software module having the same name as the software module, by performing the following steps: generating program code for an entry hook routine for performing a common entry processing function; generating program code for an exit hook routine for performing a common exit processing function; generating a return address table having addressable entries for storing the address of a calling module and hook value for the called entry point; and generating a return address function table having an exit function for each entry in the return address table.

Downs describes a method and apparatus of securely providing data to a user's system. The data is encrypted so as to only be decryptable by a data decrypting key, the data decrypting key being encrypted using a first public key, and the encrypted data being accessible to the user's system, the method comprising the steps of: transferring the encrypted data decrypting key to a clearing house that possesses a first private key, which corresponds to the first public key; decrypting the data decrypting key using the first private key; re-encrypting the data decrypting key using a second public key; transferring the re-encrypted data decrypting key to the user's system, the user's system possessing a second private key, which corresponds to the second public key; and decrypting the re-encrypted data decrypting key using the second private key.

Neither Pekowski02, Pekowski01 nor Downs taken alone or in combination teaches or suggests in a third program module associating each of a plurality of stubs respectively with each of a plurality of methods associated with the second program module, wherein each stub comprises a code segment performing a unique non-standard calling convention into the second program module, wherein each stub includes at least a first instruction to push function parameters onto the call stack, a second instruction to call an authenticator module for authenticating that a stub has not been modified and a third instruction comprising

embedded unique data for the stub, wherein the embedded data comprises a vtable entry descriptor for the desired method, corresponding to a vtable for the third module, wherein the vtable is covered and comprises a list of function pointers to functions associated with the second module arranged in a random order, the random order unique for each second module as recited in amended claim 1. In particular, none of the references disclose in any manner use of a vtable to achieve the described functionality.

Pekowski02, Pekowski01 and Downs are further deficient as they fail to teach or suggest from a first program module, issuing a first call to a stub in the third program module associated with the desired method, whereupon after the first call, the call stack comprises at least a first parameter corresponding to a return address associated with the stub, a second parameter corresponding to a parameter depth (cArgs) and a third parameter corresponding to a return address of the first program module, the first, second and third parameters arranged in a top-down order as recited in amended claim 1. None of the cited references describes a call stack structure, let alone one having the particular structure recited in amended claim 1.

Further, neither Pekowski02, Pekowski01 nor Downs taken alone or in combination teach or suggest that a third program module calls the second program module using a non-standard calling convention. Thus, since neither Pekowski02, Pekowski01nor Downs taken alone or in combination teach or suggest the cited claim limitations, claim 1 should be allowed. Claims 2 and 5-11 depend from and therefore include all the limitations of claim 1. Thus, for at least the reasons previously discussed with respect to claim 1, claims 2 and 5-11 should also be allowed.

Rejection Of Claims 12-17 Under 35 U.S.C. § 103(a)

Claims 12-17 stand rejected under 35 U.S.C. § 103(a) as being allegedly being anticipated by Pekowski02 in view of Pekowski01 and in further view of Downs. Claim 12 as amended recites a method of verifying a context in which a first program module has been called and as amended includes limitations similar to claim 1 including stubs *wherein each stub comprises a code segment performing a unique non-standard calling convention into the second program module, wherein each stub includes at least a first instruction to push function parameters onto the call stack, a second instruction to call an authenticator module for authenticating that a stub has not been modified and a third instruction comprising*

embedded unique data for the stub, wherein the embedded data comprises a vtable entry descriptor for the desired method, corresponding to a vtable for the third module, wherein the vtable is covered and comprises a list of function pointers to functions associated with the second module arranged in a random order, the random order unique for each second module.

As discussed, none of the cited references teach or suggest the use of a vtable for achieving the described functionality. Thus, for at least the reasons stated with respect to claim 1, claim 12 should be allowed. Claims 13-17 depend from and therefore include all the limitations of claim 12. Thus, for at least the reasons stated with respect to claim 12, claims 13-17 should also be allowed.

Rejection Of Claims 18 and 21-23 Under 35 U.S.C. § 103(a)

Claims 18 and 21-23 stand rejected under 35 U.S.C. § 103(a) as being allegedly being anticipated by Pekowski02 in view of Pekowski01 and in further view of Downs. Claim 18 as amended recites a program module stored in a computer-readable storage medium and as amended includes limitations similar to claim 1 including stubs *wherein each stub comprises a code segment performing a unique non-standard calling convention into the second program module, wherein each stub includes at least a first instruction to push function parameters onto the call stack, a second instruction to call an authenticator module for authenticating that a stub has not been modified and a third instruction comprising embedded unique data for the stub, wherein the embedded data comprises a vtable entry descriptor for the desired method, corresponding to a vtable for the third module, wherein the vtable is covered and comprises a list of function pointers to functions associated with the second module arranged in a random order, the random order unique for each second module.*

As discussed, none of the cited references teach or suggest the use of a vtable for achieving the described functionality. Thus, for at least the reasons stated with respect to claim 1 and claim 12, claim 18 should be allowed. Claims 21-23 depend from and therefore include all the limitations of claim 18. Thus, for at least the reasons stated with respect to claim 18, claims 21-23 should also be allowed.

Rejection Of Claim 24 Under 35 U.S.C. § 103(a)

Claim 24 stands rejected under 35 U.S.C. § 103(a) as being allegedly being anticipated by Pekowski02 in view of Pekowski01 and in further view of Downs. Claim 24 as amended recites a computer-readable storage medium storing computer-executable instructions to perform a method that facilitates verification of a call stack and recites limitations similar to claims 1, 12 and 18 of using a vtable to achieve the described functionality. Thus, for at least the reasons stated with respect to claims 1, 12 and 18, claim 24 should be allowed.

Conclusion

In view of the above amendments and remarks, applicant respectfully submits that the present invention is in condition for allowance. Reconsideration of the application is respectfully requested.

Date: November 13, 2008

/Kenneth R. Eiferman/
Kenneth R. Eiferman
Registration No. 51,647

Woodcock Washburn LLP
Cira Centre
2929 Arch Street, 12th Floor
Philadelphia, PA 19104-2891
Telephone: (215) 568-3100
Facsimile: (215) 568-3439